Reti di Calcolatori L-A
Appello del 21/12/2005
Compito 1

Cognome: ……
Nome: ……
Matricola: ……

Tempo a disposizione: 2h

E' obbligatorio mettere Cognome Nome Matricola e Numero Compito all'inizio di ogni file sorgente, pena la non valutazione del compito, che viene stampato in modo automatico, solo in caso siano presenti gli elementi detti sopra.

Si devono consegnare **tutti i file sorgente e tutti gli eseguibili prodotti singolarmente** (*per favore, solo quelli relativi ai file sorgente consegnati!!!*).

La prova intende valutare le capacità progettuali e di programmazione sia in **ambiente Java** che in **ambiente C**, pertanto è consigliabile sviluppare, *almeno in parte*, **entrambe** le soluzioni richieste.
In entrambi gli esercizi, sia in Java che in C, si effettuino gli opportuni controlli sui parametri della richiesta e si gestiscano le eccezioni, tenendo presente i criteri secondo cui si possa ripristinare il funzionamento del programma oppure si debba forzarne la terminazione.

Leggete con attenzione le specifiche del problema prima di impegnarvi "a testa bassa" nello sviluppo delle singole parti.
Naturalmente, i componenti da consegnare devono essere stati provati.

****

Si richiede il progetto della gestione dei servizi **Parla**, per utenti che vogliono comunicare attraverso l'ambiente realizzato. Obiettivo è l'insieme delle funzionalità di supporto per il provider dei servizi e non la realizzazione delle funzioni finali di scambio messaggi.
I servizi di Parla sono organizzati in **stanze** (con nomi che rappresentano l'argomento della stanza) e, per ciascuna stanza, si prevedono al massimo un certo numero di **utenti**. Ogni stanza prevede due modalità: scambi **broadcast di stanza o multicast** (**M**), e scambi **punto-a-punto** (**P**) tra utenti. Un utente si può iscrivere ad una o più stanze e poi può inviare e ricevere i messaggi in transito sulle stanze, senza stato permanente dei messaggi stessi.
Si vogliono realizzare le funzionalità di gestione:

1. **aggiunta di una stanza**: questa operazione richiede il nome delle stanza e il tipo di comunicazione da instaurare, e inizializza la prima stanza libera disponibile nella struttura dati;
2. **eliminazione di un utente**: questa operazione richiede il nome dell'utente, scorre le stanze liberandole dall'utente, e restituisce la lista di tutte (e sole) le stanze liberate;
3. **visualizzazione dello stato delle stanze**: questa operazione visualizza l'attuale stato di tutte le stanze, indicando anche, per ogni stanza, il tipo di operatività;
4. **sospensione del lavoro in una stanza**: questa operazione richiede il nome di una stanza e ne sospende la operatività producendo un cambiamento di stato della stanza stessa (stato **S**, aggiunto a P o M).

Si progetti con particolare attenzione la **struttura dati** che mantiene lo stato delle N stanze (L, per libero a default) per al massimo di K utenti, da implementare opportunamente nei diversi ambienti richiesti, Java e C. Ogni stanza prevede modalità o broadcast o punto-a-punto, e sospeso, e registra gli utenti presenti.

Nome	Stato	Utente 1	Utente 2	Utente 3	Utente 4	...	Utente K-1	Utente K
Stanza 1	**P**	**Pippo**	**Minnie**	**Orazio**	**L**	**...**	**L**	**L**
Stanza 2	**SP**	**Pippo**	**Pluto**	**L**	**L**	**...**	**L**	**L**
Stanza 3	**M**	**Pluto**	**Paperino**	**Quo**	**Qui**	**...**	**L**	**L**
L	**L**	**L**	**L**	**L**	**L**	**...**	**L**	**L**
Stanza N	**SM**	**L**	**L**	**L**	**L**	**...**	**L**	**L**

Parte Java

Utilizzando java RMI sviluppare un'applicazione C/S che consenta di effettuare le operazioni remote per:
- aggiungere una stanza alla struttura dati;
- eliminare un utente dalla struttura dati.

Il progetto RMI si basa su:
- un'interfaccia (***RemOp,*** contenuta nel file *RemOp.java*) in cui vengono definiti i metodi invocabili in remoto dal client:

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RemOp extends Remote {
   boolean aggiungi_stanza(String nomeStanza, char tipoComunicazione)
                                                    throws RemoteException;
   Stanza[] elimina_utente(String nomeUtente) throws RemoteException;
}
```

Il metodo **`aggiungi_stanza`** restituisce valore logico *vero* in caso di successo, *falso* in caso d'errore, ad esempio, se esiste già una stanza con lo stesso nome nella struttura dati o se la struttura dati è piena.
Il metodo **`elimina_utente`** restituisce la lista delle stanze liberate, *null* in caso d'errore, ad esempio, se l'utente non è presente in nessuna delle stanze mantenute nella struttura dati.

Si progettino inoltre le classi:
- ***Server*** (contenuta nel file *Server.java*), che implementa i metodi del server invocabili in remoto;
- ***Client*** (contenuta nel file *Client.java*), che realizza l'interazione con l'utente proponendo ciclicamente i servizi che utilizzano i due metodi remoti, e stampa a video i risultati, fino alla fine del file di input da tastiera.

Parte C

Sviluppare un'applicazione C/S che consenta di effettuare le operazioni remote per:
- visualizzare lo stato attuale di tutte le stanze;
- sospendere l'operatività di una stanza;

con un servitore multiservizio (uso di select).

Più in dettaglio:
- il **client_stream** realizza la funzionalità di visualizzazione dello stato delle stanze utilizzando **socket stream**. Il client propone ciclicamente all'utente l'operazione, fino alla fine del file di input da tastiera, dopodichè libera opportunamente le risorse e termina. Ad ogni richiesta, il client riceve e stampa a video la struttura dati.

- Il **client_datagram** realizza la funzionalità di sospensione dell'operatività di una stanza utilizzando **una socket datagram**. Il client chiede ciclicamente all'utente il nome della stanza da sospendere, fino alla fine del file di input da tastiera, e, al termine, libera opportunamente le risorse e termina.
  Ad ogni richiesta dell'utente il client invia la richiesta, riceve l'esito dell'operazione, e lo stampa a video.

- Il **server** principale unico discrimina le richieste utilizzando la primitiva select. Il server gestisce in modo parallelo la funzionalità di visualizzazione dello stato attuale delle stanze, mentre la funzionalità per sospendere l'operatività di una stanza può essere realizzata in modo seriale o parallelo.
  Per ogni richiesta di **visualizzazione dello stato** il server genera un figlio che gestisce l'interazione col client fino alla chiusura della connessione da parte del client. Per ciascuna richiesta il figlio legge e invia la struttura dati.
  Per ogni richiesta di **sospensione dell'operatività**, il server riceve il nome della stanza, ne cambia lo stato e invia la risposta al client: 0 in caso di successo, -1 in caso d'errore, ad esempio, se lo stato della stanza era già sospeso.